COLLABORA

Image Signal Processor (ISP) Drivers
& How to merge one upstream

Helen Koike
Senior Software Engineer

Embedded Linux Conference
North America

About me

- @ Collabora since 2016
- Mostly working on the kernel – media subsystem:
  - Maintainer of rkisp1 driver
  - Maintainer of vimc driver
- Outreachy intern in 2015 – vimc projet
- Co-coordinator of Linux Kernel project in Outreachy

Main goal of this presentation

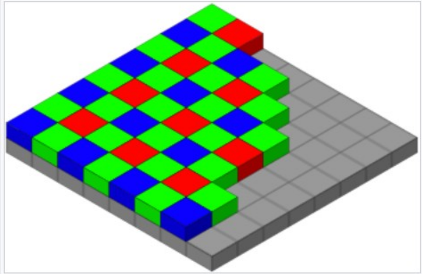
- Overview of Camera→ISP→Memory pipeline
- Overview of Media Framework
- Design choices when implementing a driver
- Lessons learned when upstreaming rkisp1 driver
- Userspace tools (libcamera)

COLLABORA

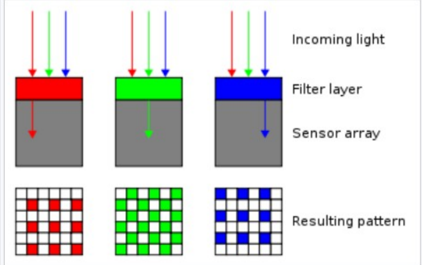
Camera→ISP→Memory

Camera sensor

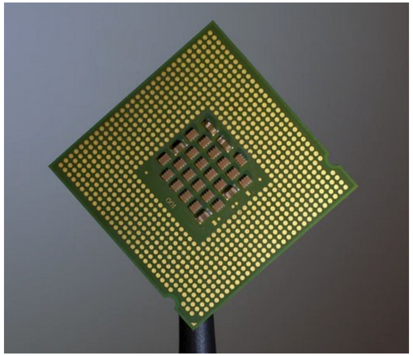
Application



The Bayer arrangement of color filters on the pixel array of an image sensor



Profile/cross-section of sensor



What is an ISP?

- Image signal processor
- Common use case:
  - ISP receives the reading all those small color sensors
  - Transforms in an image usable for userspace
- Performs several other image transformations

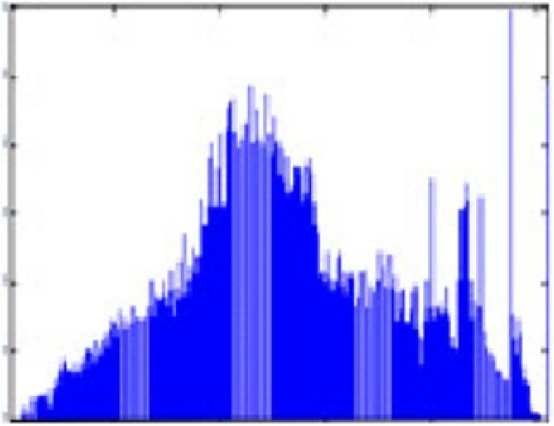
Image Processing

- Format conversion (debayering, RGB, YUV)
- Crop / Resize
- White balance
- Compose
- Image stabilization
- Effects / filters
- Flip / Rotate
- etc

Hardware accelerated image processing

Offloads the CPU

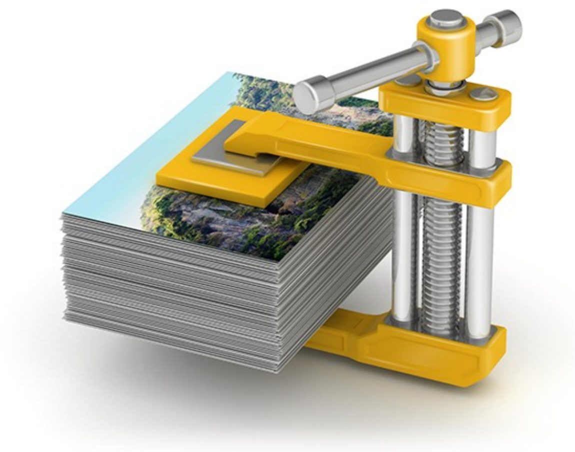
Statistics



- ISP can generate statistics:
  - Histograms
  - Area contrast
  - etc
- Used by userspace to implement algorithms such as:
  - Histogram equalization
  - 3A (auto-focus, auto-exposure, auto-white balance)

What an ISP is not

- ISP is not a codec
- ISPs work with raw/uncompressed images
- Codecs:
  - Encoders: raw image → compressed image format (such as H.264, JPEG, VP9)
  - Decoders: compressed image → raw image



ISPs architecture

Inline vs Offline

Offline

- 2 phases:
  - Sensor → Memory
  - Memory → ISP → Memory
- Usually implemented in two separate drivers
  - Coordinated by userspace
  - Example Intel IPU3:

    IPU3 CIO2 (camera interface) driver: gets the image from the sensor

    IPU3 ImgU driver: process this image and sends to userspace

Inline

- Data reaches memory only in the end:
  - Sensor → ISP → Memory

- Example: rkisp1 driver

Hybrid

- Can get the image directly from the sensor or from memory
- Can behave as inline, or perform the second phase of offline
- Ex: MT8183 P1

```
                  +------------+
                  |    DMA     |---------->|\
                  |   Input    |           | \          +----------+          +------------+
                  +------------+           |  |         |  Bayer   |          | Processed  |
                                           |MUX|-------->|Processing|--------->|  Output    |
+------------+    +------------+           |  |         +----------+          +------------+
|   Sensor   |    |   Camera   |           | /
|            |--->|  Interface |---------->|/
+------------+    +------------+
```

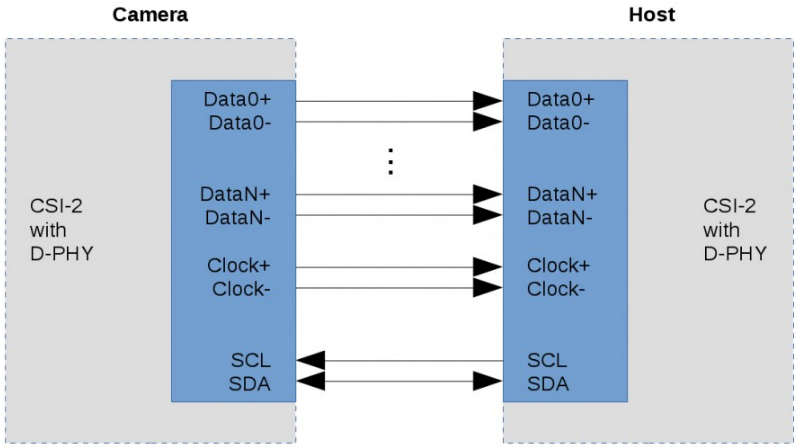
COLLABORA

MIPI DPHY (quick overview)

Bus – MIPI DPHY

- Very common bus used in the market for cameras and displays
- Specified by MIPI Alliance
- Physical layer with high data-rate
- 4k images with a good frame rate

Bus - MIPI DPHY



- Up to 4 data lanes
- I2C bus for configuration

- On top of this bus there can be two protocols:
  - MIPI DSI-2: Display Serial Interface, to output images
  - MIPI CSI-2: Camera Serial Interface, to capture images

- MIPI DPHY/CSI-2 → frequent term in ISP land

COLLABORA

Study case - RKISP1

Rockchip RK3399 ISP

- rkisp1 is the driver of the ISP block present in Rockchip RK3399 SoCs
- RK3399 SoC can be found in devices such as:
  - Scarlet Chromebooks
  - RockPi boards
  - Pinebook Pro laptops

Rockchip RK3399 ISP

- Originally written by Rockchip
- Merged in kernel 5.6
- drivers/staging/
- 9k+ lines of code

Rkisp1 hw architecture

```
                                                            rkisp1-resizer.c          rkisp1-capture.c
                                                         |====================|  |=======================|
                               rkisp1-isp.c                              Main Picture Path
                       |==========================|      |===============================================|
                       +-----------+  +--+--+--+--+      +--------+  +--------+              +-----------+
                       |           |  |  |  |  |  |      |        |  |        |              |           |
+--------+    |\       |           |  |  |  |  |  |   -->|  Crop  |->|  RSZ   |------------->|           |
|  MIPI  |--->|  \     |           |  |  |  |  |  |   |  |        |  |        |              |           |
+--------+    |   |    |           |  |IE|IE|IE|IE|   |  +--------+  +--------+              |  Memory   |
              |MUX|--->|     ISP   |->|0 |1 |2 |3 |---+                                      | Interface |
+--------+    |   |    |           |  |  |  |  |  |   |  +--------+  +--------+  +--------+  |           |
|Parallel|--->|  /     |           |  |  |  |  |  |   |  |        |  |        |  |        |  |           |
+--------+    |/       |           |  |  |  |  |  |   -->|  Crop  |->|  RSZ   |->|  RGB   |->|           |
                       |           |  |  |  |  |  |      |        |  |        |  | Rotate |  |           |
                       +-----------+  +--+--+--+--+      +--------+  +--------+  +--------+  +-----------+
                                                         |===============================================|
                                                                         Self Picture Path
```

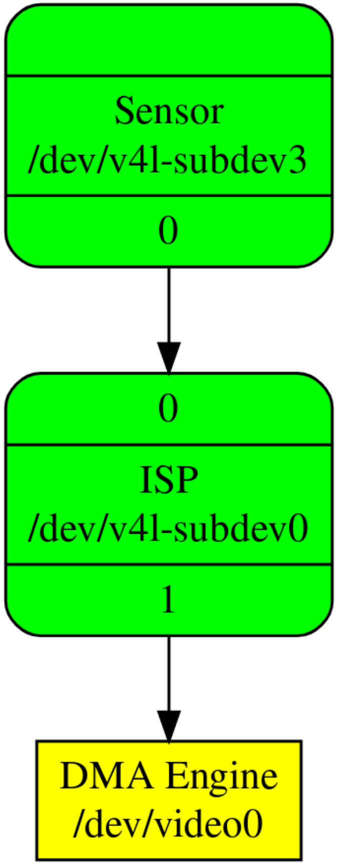
Rkisp1 hw architecture

- ISP Comprises with:
  - Image Signal Processing
  - Many Image Enhancement Blocks
  - Crop
  - Resizer
  - RBG display ready image
  - Image Rotation
- Self-path: preview
- Main-path: picture

Kernel media framework

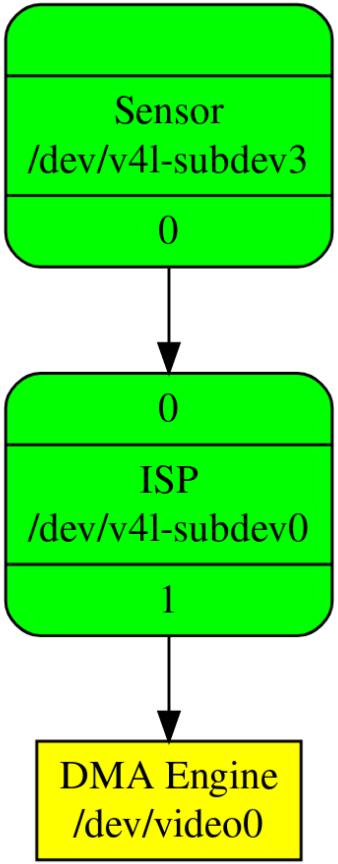
Media topology

- Linux kernel exposes a topology to userspace
- Userpace can query /dev/mediaX
  - Retrieve how inner blocks are interconnected
  - Order of image processing

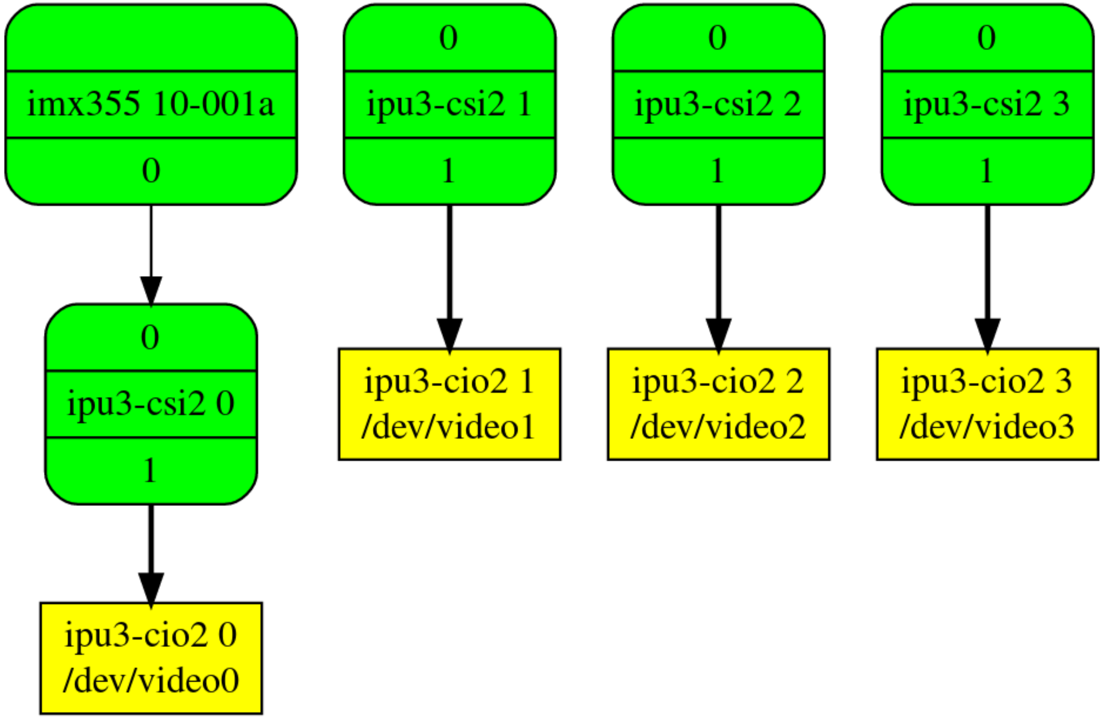


Media topology

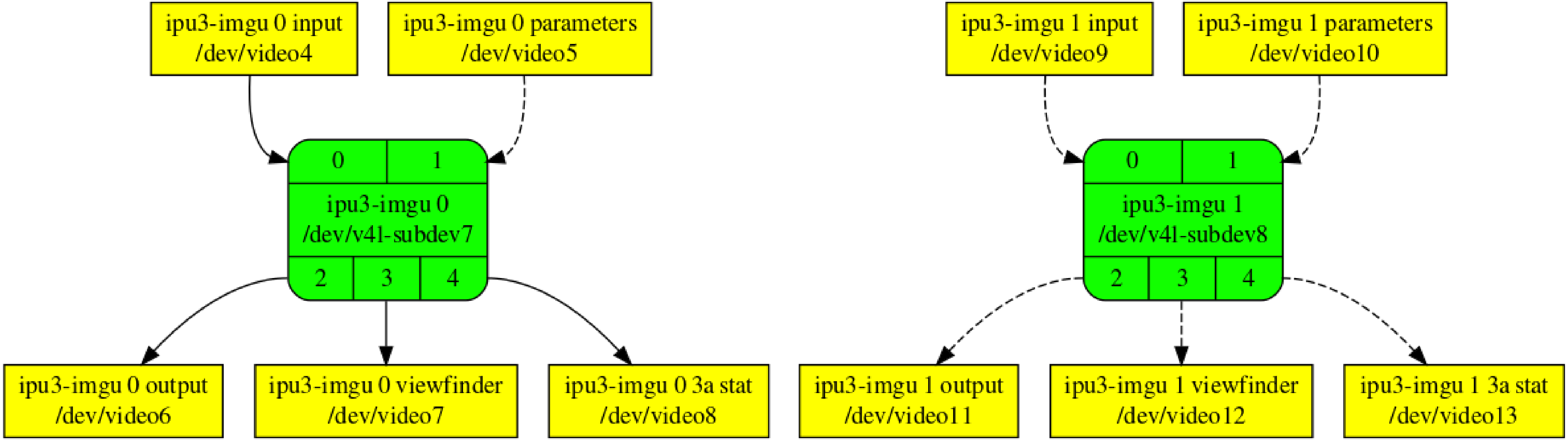
- Two types of nodes:
  - **subdevices**: inner parts of the hardware
  - **video devices**: dma engine, where userspace queues and dequeues buffers, containing images or metadata to/from the hardware
- Connected by links between pads
- NOTE: sensor is usually a separated driver



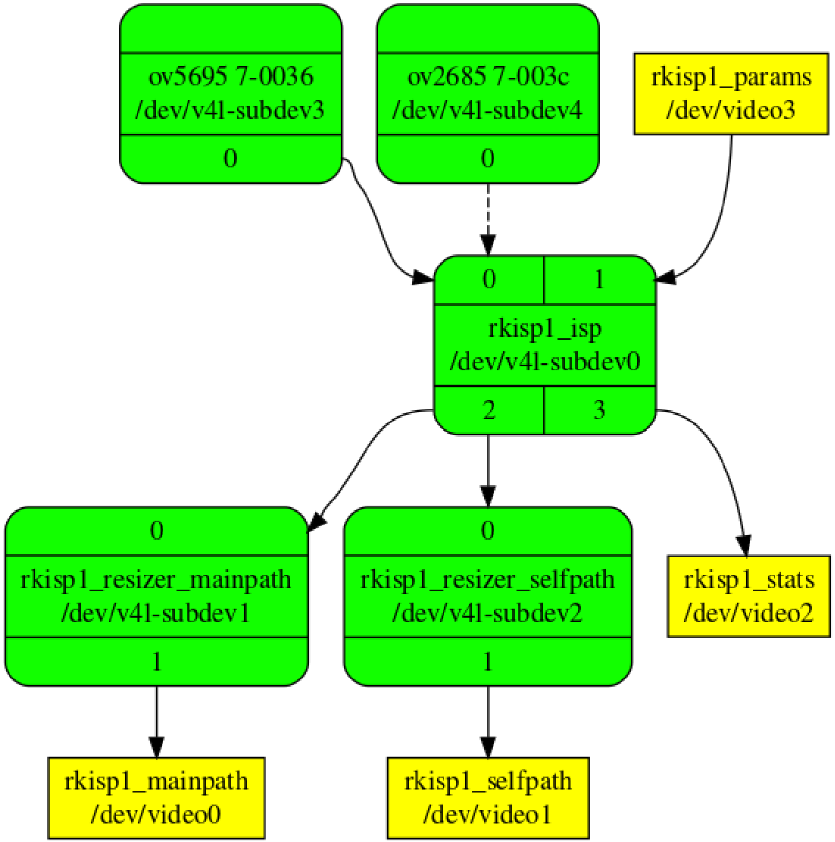
IPU3 CIO2 – offline – 1st phase



IPU3 ImgU – offline – 2nd phase



RKISP1 - inline

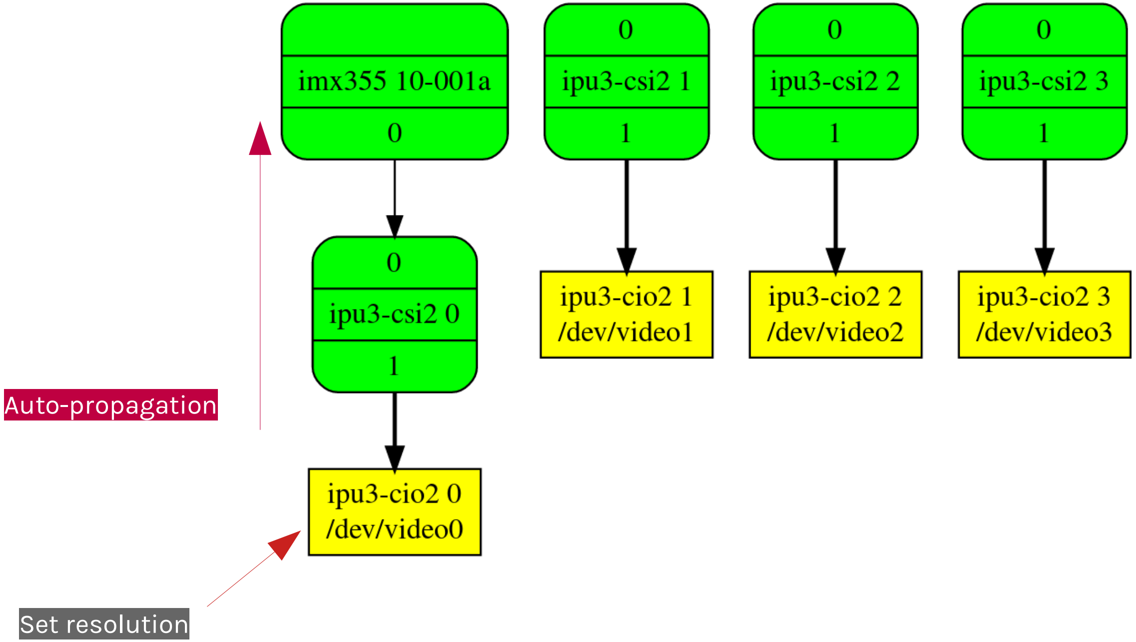


COLLABORA

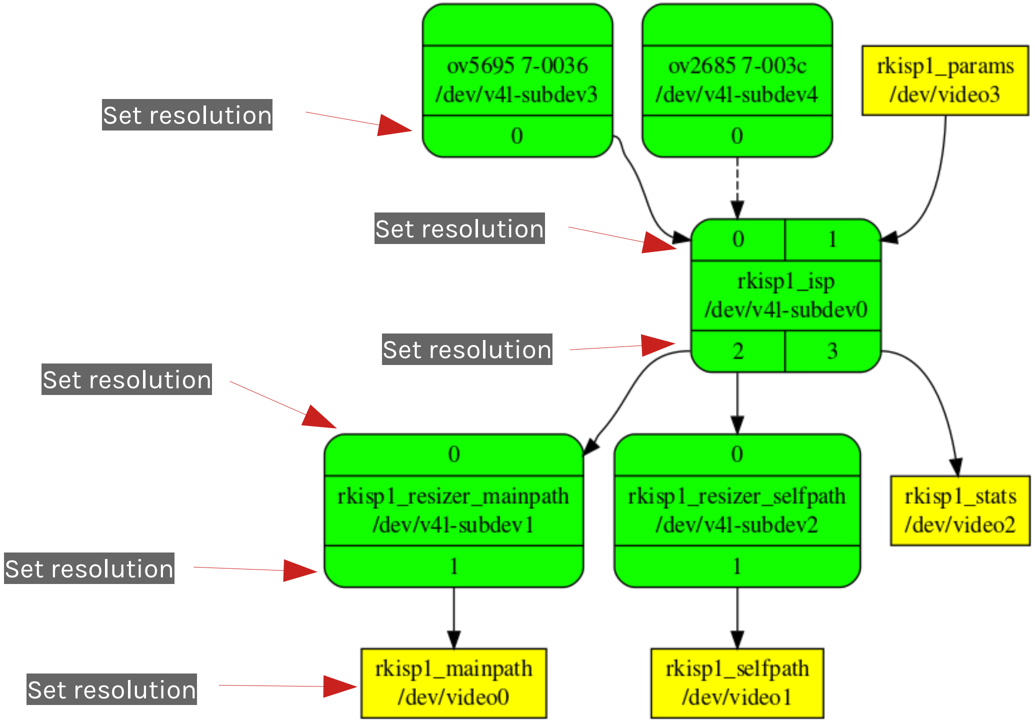
Driver config architecture

Auto vs Manual config propagation

Auto config propagation



Manual config propagation

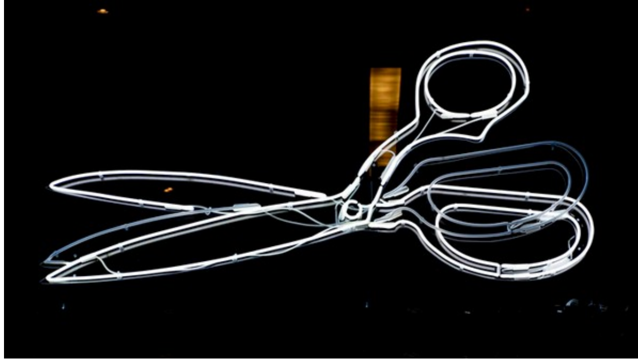


Manual config propagation

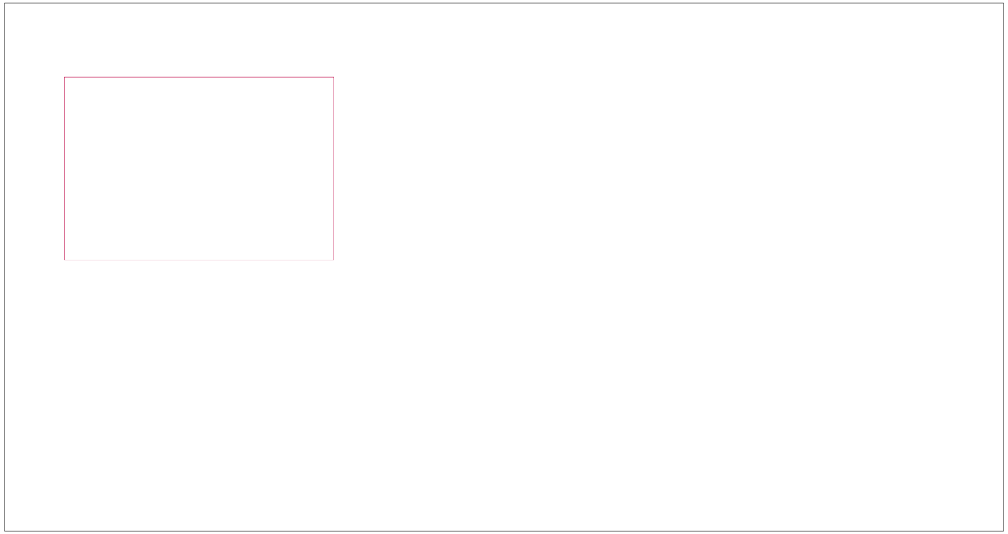
- Increases complexity for userspace
- If formats don't match → fail on STREAMON
- Finer grain configuration in inner blocks of the hardware
- More blocks exposed, more complex
- Extendable

Why rkisp1 is manual?

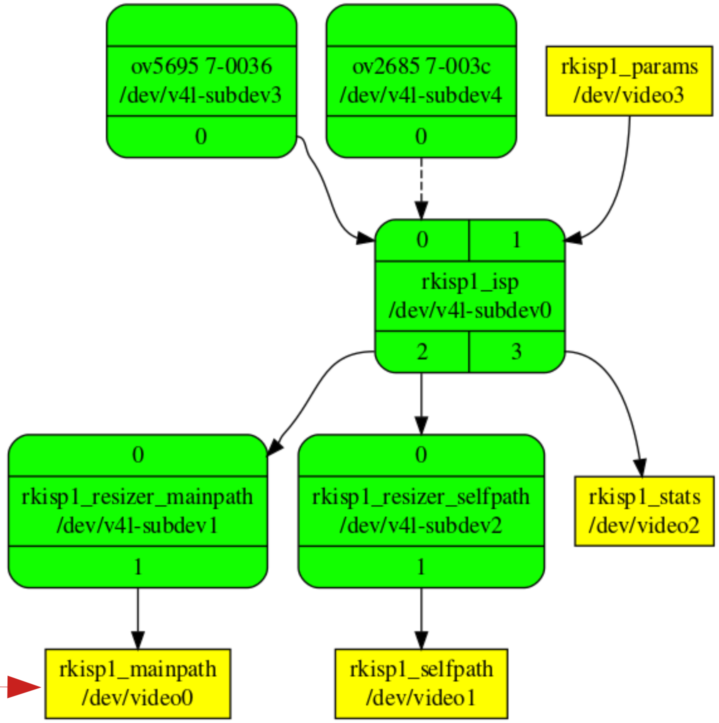
Crop



- Specify a sub-rectangle in the image



Crop - rkisp1



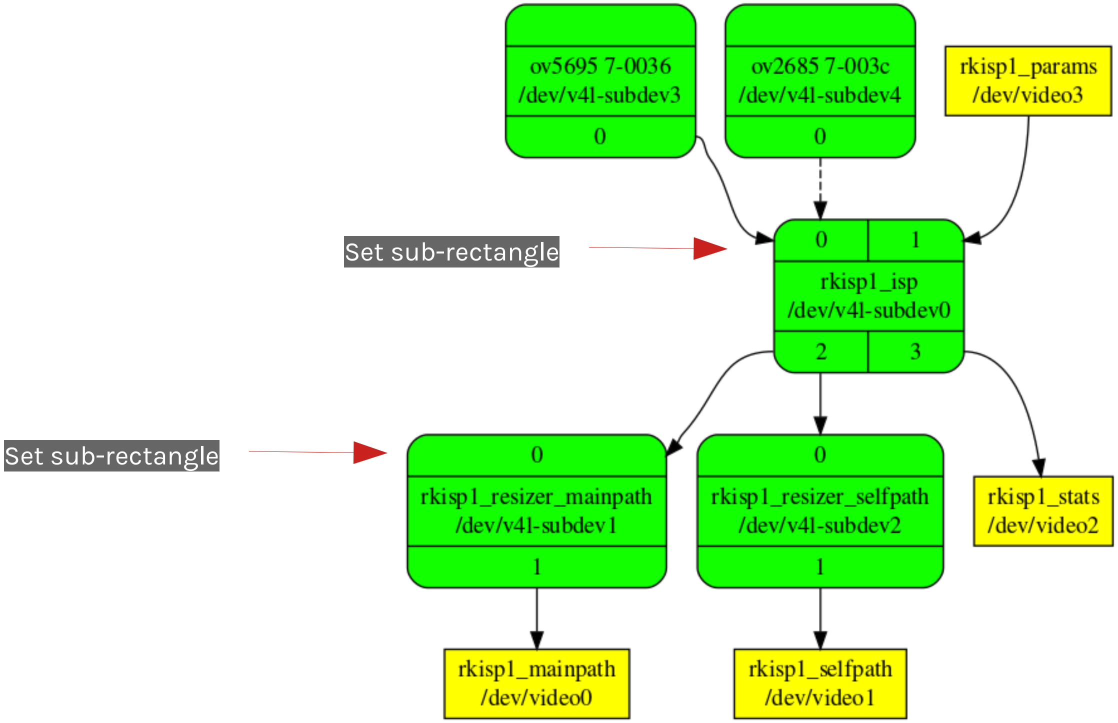
Set sub-rectangle?

Crop - rkisp1

- rkisp1 allows cropping the image from the sensor
- rkisp1 allows cropping the image before resizing

- Exposing crop once in the video node would be confusing

Crop - rkisp1

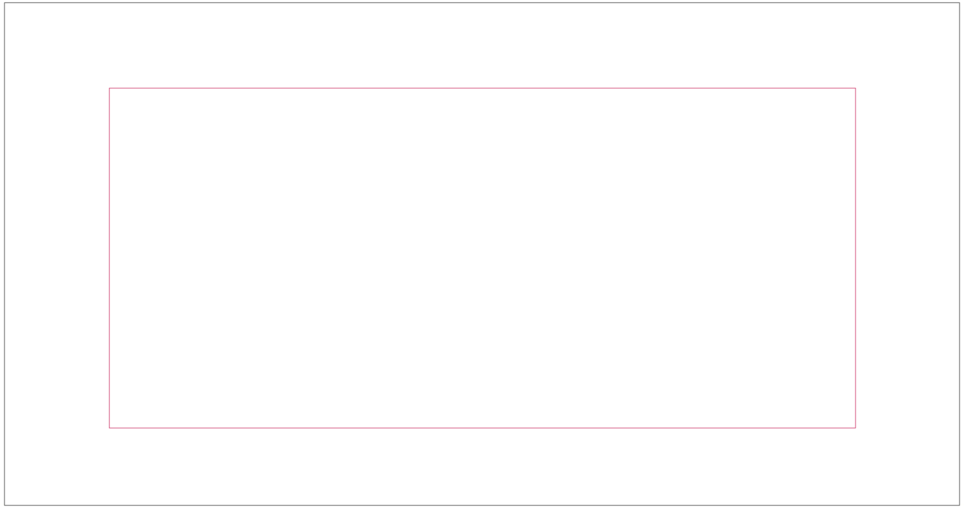


COLLABORA

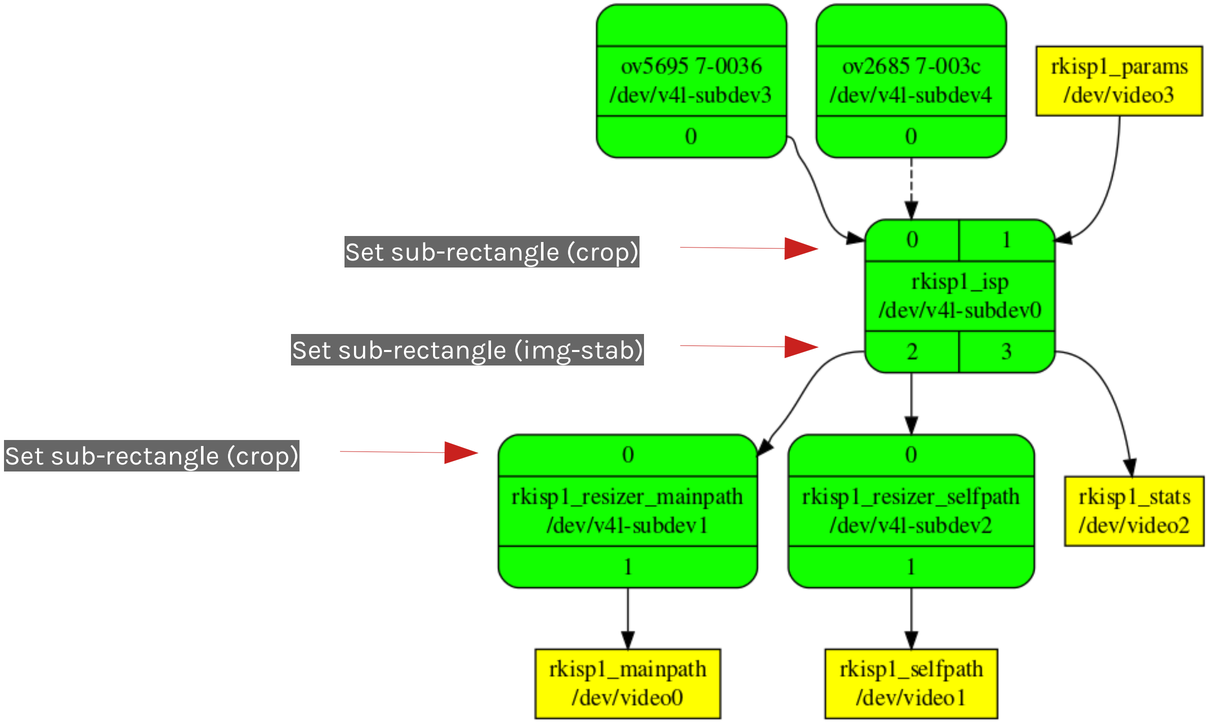
Open First

Image stabilizer

- “Lock” sub-rectangle in the picture
- Shaking the phone won’t shake the image much



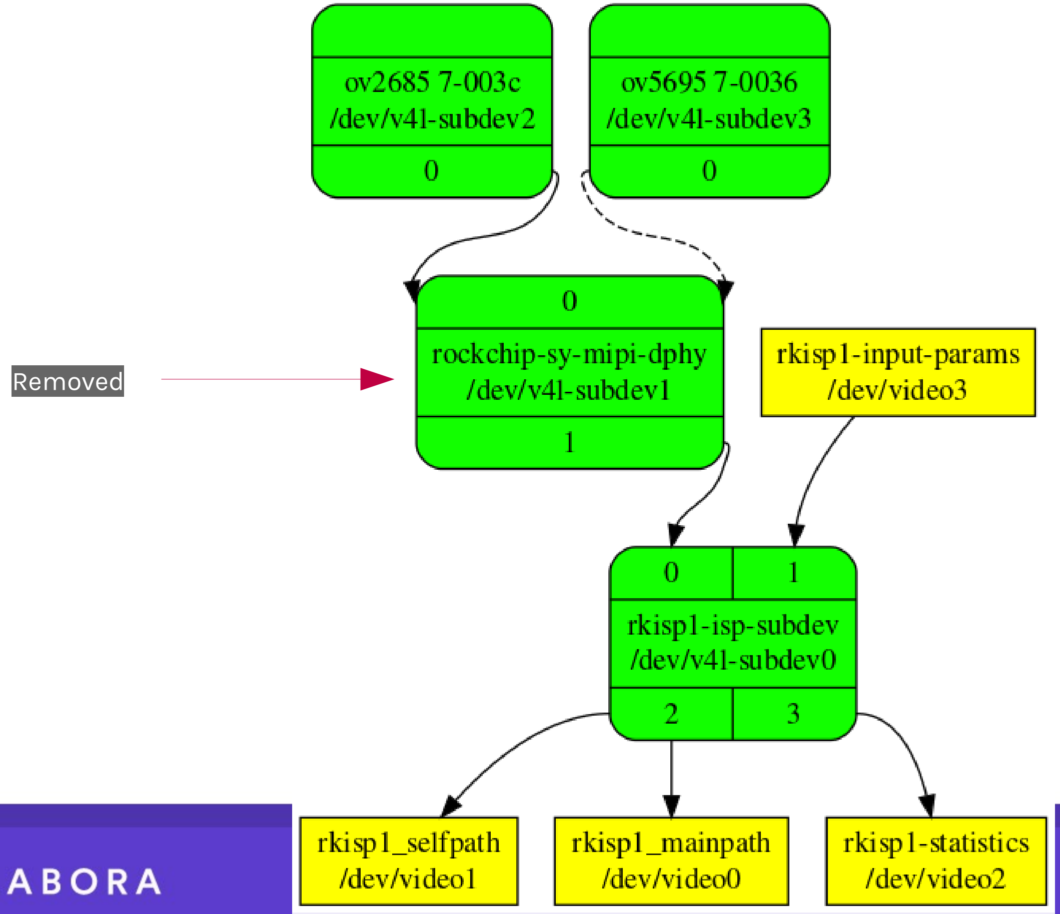
Setting sub-rectangles



COLLABORA

Phy subsystem

Rkisp1 – original topology



Phy Abstraction Layer

- Manual config propagation → more subdevices, more complex for userspace
- Re-think exposed blocks
- Phy block → no image configuration exposed
- Topology → image processing steps
- Same processing steps can be used on top of different buses
  - ex. rkisp1: parallel (not implemented), MIPI-DPHY/CSI2

Phy – Lessons learned

- Lessons learned:
  - Migrate bus code to PHY Abstraction Layer (drivers/phy/)
  - Generic topology for any bus - less complex for userspace
  - ISP driver is much cleaner
  - Phy driver can be used for DSI

Lessons learned

Updating to staging

- V4L2 community is open to accept drivers in staging
  (with the condition that you work on it to move it out asap)
- Detailed TODO list
- Make it available to other people to use
- Improve workflow, easier to get contributions from others, testing, bug reports
- Decrease maintenance cost → no need to keep rebasing

More lessons learned

- Don't be afraid to re-organize the code (files, namings, code order, re-writing functions)
- Split the code between different files per implementation node, at least between video nodes and subdevice nodes
- Separate the code that configures the hardware, from the code that deals with the V4L2 API
- Remove code you are not using, you that you can't test, for example:
  - rk3288 support
  - phy driver ports (SoC has 2 MIPI-DPHY/CSI2 ports, I had was only using one)
  - Simplify the code – but keep extendable
  - Lots of macros in headers

Userspace support

Libcamera

Complex topologies

- Not all features are auto discoverable

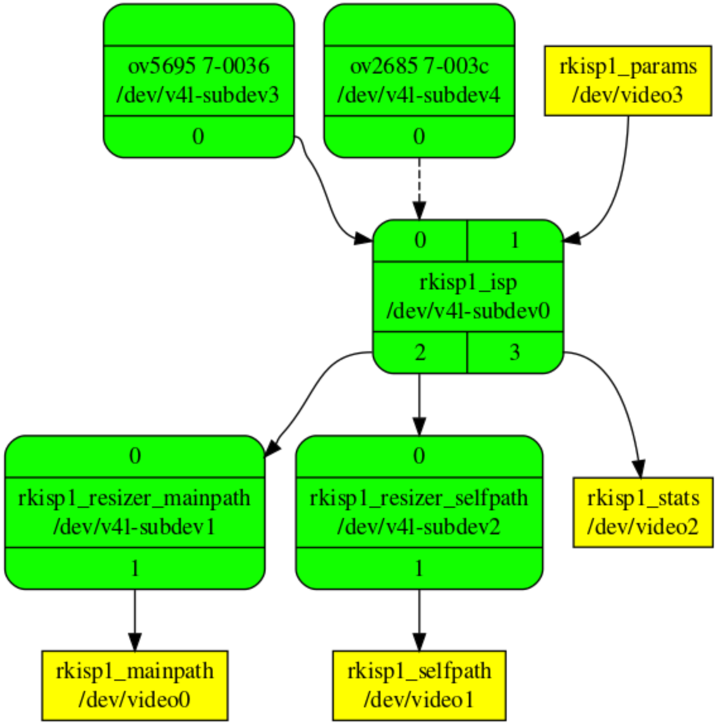
  Examples (rkisp1):

  - sub-rectangle for cropping

    vs sub-rectangle for image stabilizer

  - Meta-data buffers structure:

    - rkisp1_stats
    - rkisp1_params



Complex topologies

- Requires userspace specific implementation for specific drivers
- Specific applications to specific hardware
- Not very reusable code
- Hard to test

Libcamera

- Open source camera stack for many platforms with a core userspace library
- Userspace drivers
- Image processing algorithms

Architecture

```
--------------------------< libcamera Public API >---------------------------
              ^                                      ^
              |                                      |
              v                                      v
       +-------------+  +-------------------------------------------------+
       |   Camera    |  |  Camera Device                                  |
       |   Devices   |  | +---------------------------------------------+ |
       |   Manager   |  | | Device-Agnostic                             | |
       +-------------+  | |                                             | |
              ^         | |                    +------------------------+ |
              |         | |                    |   ~~~~~~~~~~~~~~~~~~~~~  |
              |         | |                    |  {  +---------------+  } |
              |         | |                    |  }  | ////Image//// |  { |
              |         | |                    | <-> | /Processing// |  } |
              |         | |                    |  }  | /Algorithms// |  { |
              |         | |                    |  {  +---------------+  } |
              |         | |                    |   ~~~~~~~~~~~~~~~~~~~~~  |
              |         | |                    | ======================== |
              |         | |                    |     +---------------+    |
              |         | |                    |     | //Pipeline/// |    |
              |         | |                    | <-> | ///Handler/// |    |
              |         | |                    |     | ///////////// |    |
              |         | +--------------------+     +---------------+    |
              |         |                                 Device-Specific |
              |         +-------------------------------------------------+
              |                     ^                        ^
              |                     |                        |
              v                     v                        v
       +--------------------------------------------------------------------+
       | Helpers and Support Classes                                        |
       | +-------------+  +-------------+  +-------------+  +-------------+ |
       | |  MC & V4L2  |  |   Buffers   |  | Sandboxing  |  |   Plugins   | |
       | |   Support   |  |  Allocator  |  |     IPC     |  |   Manager   | |
       | +-------------+  +-------------+  +-------------+  +-------------+ |
       | +-------------+  +-------------+                                   |
       | |  Pipeline   |  |     ...     |                                   |
       | |   Runner    |  |             |                                   |
       | +-------------+  +-------------+                                   |
       +--------------------------------------------------------------------+

         /// Device-Specific Components
         ~~~ Sandboxing
```

Tips

- Add/push/update support for your hardware in Libcamera
- Easier to test
- More users
- More developers involved
- Contribute with the project

Thank you!